Efficient String Matching Using Deterministic Finite Automation Hardware: Speed vs Area Tradeoff

AakankshaPandey
Department of CSE & IT
MANIT, Bhopal.

NilayKhare , PhD
Department of CSE & IT
MANIT, Bhopal.

ABSTRACT
Pattern matching is a crucial task in several critical network services such as intrusion detection and matching of the IP address during packet forwarding by the router. In this paper we present an speed vs area tradeoff of the the original DFA and the DFA called delayed input DFA(D^2FA) with optimized area by eliminating the redundant transition edges.In delayed input DFA the area required to store transition table reduces to 60% of the original DFA but the clock pulse required to execute the process increases almost 40% of the original DFA. The comparison of area and speed is presented. This area optimized architecture of DFA is simulated and synthesized using VHDL on the Xilinx ISE 12.4.

Keywords:
String Matching,DFA, VHDL.

1. INTRODUCTION
With the increased amount of data transferred by network the amount of malicious packet also increased therefore it is necessary to protect the network against malicious attack. Intrusion Detection Systems (IDS) are emerging as one of the most promising way of providing protection to systems on the network against these malicious attacks. Intrusion Detection System continuously monitors the network traffic for suspicious pattern and informs the administrator to take proper action. String matching is the heart of IDS. String matching matches each incoming packet against some stored patterns and identify the suspicious activity. The pattern matching is also used by the router to identify the appropriate outgoing line of the packet during packet transmission .The pattern is matched against the text string. Suppose given text string T = t1t2 …. tn and pattern (keyword)P = p1p2 ….pm, verify if string P is a substring of text T. This task is very simple but it is used very frequently in case of networks application. Very fast algorithms are therefore necessary for this task.

The pattern matching can be implemented in both software and hardware. The main motivation of implementing it into the
hardware is the performance gap. Hardware provides efficientand flexible way of implementation. FPGA (field Programmable Gate Array) provides flexibility and FPGAincrease theperformance of software based system by 600x for large patterns.

There are several techniques that are exist for pattern matching in hardware like comparator based architecture in which discrete comparator are used to match particular character similarly hash based function is also used which uses the digest of the pattern for matching purpose. Finite automata are also used for this purpose. This paper mainly focuses on finite automata based pattern matching. Finiteautomata are very useful way for understanding andsolving manytext processing problems. Deterministic FiniteAutomata (DFA) is widely used in existing work toaccelerate regular expression matching. A deterministic finite state automaton (DFA) is a simple language recognition device. Finite automata provide the easiest way of pattern matching but Depending on the application being considered, it can be the case that the size of the input string tothe DFA is large (e.g. the text T in secure pattern matching), or the size of the DFA itself (e.g. whenmany patterns are combined into one DFA). Therefore, it needs to ensure efficiency and scalability when run on large DFAs and/or input strings.

In this paper we present the area vs speed tradeoff of the Deterministic Finite Automata (DFA) and the delayed input Deterministic Finite automata(D^2FA).We will see how area occupied by the state transition table drastically get decreased by eliminating redundant transition in D^2FAalong with how the time taken by the input pattern for processing get increased in detail.

The rest of the paper is organized as follows section I describes the introduction part of the pattern matching and its application in the network ,section II describe the related work in this field and section III presents the background information of the work, next section deals with the implementation result and the comparison of speed and area taken by the normal DFA and delayed input DFA and last section is the conclusion part.

2. RELATED WORK
In the past few years numerous hardware based pattern matching solution have been proposed.The main techniques are CAM(based architecture [5,9,13] uses discrete comparator results higher throughput with increased area and low efficiency,hash function[2,6,12,13] used to compress the string set find probable match and reduce the total number of comparison. regular expression and finite automata based patternmatching[1,2,3,4,5,11,14] results low throughput with increase the area of implementation .the main aim of this paper is to reduce the area of implementation and resource used by applying the state minimization algorithm.

Ioannis et al[10] has given the CAM based architecture uses discrete comparator for pattern matching in which the frequency of the pattern matched get increased but the area required to implement the model increases with number of patterns so they uses decoded CAM architecture for better performance and to reduce area density and pipelined Cam to increase processing speed they conclude that pipelined DCAM is the best choice for hardware implementation of pattern matching.

Recently Dhanpriya et al[11] have designed word split hash algorithm in which on the basis of sub hash the pattern is matched .So the malicious packet is detected at the initial stage if so.This architecture reduces the total number of comparison and also reduces the execution time.
Sidhu and Prasanna[14] mapped the NFA into an FPGA results the modest throughput with large area so Karuppiah and Rajaram[1] recently mapped the regular expression into DFA which reduces the number of states used results the area efficiency.

Sailesh et al [14] have given the architecture of D^2FA in which the redundant edges are removed and replaced by the single default edge. This paper deals with the comparision of the original DFA and the D^2FA time and speed. Although the required area of the state table reduces but the time is also a very important factor in network environment . So the contribution of this paper is will help to identify the good technique for the pattern matching in the application over the network.

3. BACKGROUND

3.1 Regular Expression

It is the most common way to represent the pattern to match. Full regular expressions are composed of two types of characters.The special characters (like the * from the filename analogy) are called metacharacters**,** while everything else are called literal.Literal text acting as the words and metacharacters as the grammar. The words arecombined with grammar according to a set of rules to create an expression which generate patterns. Some metacharacters are *,+,?,|,Repetition is specified with *, for zero or more, +, for one or more, and ?, forzero or one, Alternation is specified with |. In regular expression if $\sum$ is an alphabet, then $\sum^*$ denotes the set of all finite strings of symbolsin $\sum$. Any subset of $\sum^*$is a languageover $\sum$.Example of regular expression is

{^ (yes|YES|Yes)$}

This matches exactly “yes”, “Yes”, or “YES”.
Regular expressions have been used in a variety of practical applicationsto specify regular languages in a perspicuous way. The problem of decidingwhether a given string belongs to the language denoted by a particular regular expression can be implemented efficiently using finite automata. A regular expression is used for pattern matching that matches one or more string of characters. Regular expression is generated for every string in the rule set and nondeterministic / deterministic finite automata are generated that examines the one byte input at a time.

3.2 Nondeterministic Finite Automata

An NFA is represented formally by a 5-tuple, $(Q, \Sigma, \Delta, q_0, F)$, consisting of a finite set of states Q ,a finite set of input symbols Σ,a transition relation $\Delta : Q \times \Sigma \rightarrow P(Q)$,an initial (or start) state $q_0 \in Q$,a set of states F distinguished as accepting (or final) states$F \subseteq Q$.Here, P(Q) denotes the power set of Q. Let $w = a_1a_2 \ldots a_n$ be a word over the alphabet Σ. The automaton M accepts the word w if a sequence of states, $r_0, r_1, \ldots, r_n$, exists in Q with the following conditions:$r_0 = q_0, r_{i+1} \in \Delta(r_i, a_{i+1})$, for $i = 0, \ldots, n-1, r_n \in F$.

3.3 Deterministic Finite Automata:

A deterministic finite automata is similar to the Non Deterministic finite automata the only difference is in transition function ($\delta : Q \times \Sigma \rightarrow Q$) where Q is the only one state instead of power set of Q.Let $w = a_1a_2 \ldots a_n$ be a string over the alphabet Σ. The automata M accepts the string w if a sequence of states, $r_0, r_1, \ldots, r_n$,exists in Q with the following conditions:$r_0 = q_0, r_{i+1} = \delta(r_i, a_{i+1})$(for $i = 0, \ldots, n-1$)and $r_n \in F$.

DFA differ substantially from NFA in the way they process data.An essential property of DFA is that at any given point of time only one state is active ie for each input symbol a single state needs to be processed .In contrast , an NFA can have multiple active states at the same time which all need to be processed when the next input symbol is read.

3.4Delayed Input DFA(D^2FA):

The space taken by any DFA is determined by the number of states* number of transition edges and time taken by the DFA is the time taken to process the input or the number of transition that it needs to take to reach to the final state. For an ASCII alphabet, there can be upto 256 edges leaving each state, it makes the required space very high so it needs the use of some compression technique. Sailesh et al [14] have given the concept of D^2FA in which the redundant edges areeliminated. Here we will see the algorithm for the delayed input DFA(3.4.2).

3.4.1 Lemma for the elimination of redundant edges:

Lemma 1. Consider a D^2FA with distinct states u and v, where u has a transition labeled by the symbol a, and no outgoing default transition. If $\delta(u,a)=\delta(v,a)$, then theD^2FA obtained by introducing a default transition from u to v and removing the transition from u to $\delta(u,a)$ is equivalent to the original DFA.

3.4.2 Algorithm

The algorithm for constructing the Delayed input DFA is given below considering the ASCII alphabet as set of input .

ProcedureDefine_DDFA
(1)Set state; set literal[255];
(2)default={};
(3)for each state[u,v]
(4) forinteger i=0 to 255
(5) Do a=literal[i]
(6) if$\delta(u,a)= \delta(v,a)$
(7) then default={v}
{make a default transition
from v to u}
(8)fi;
(9)oD;
(10)rof;
(11)rof;
End

4. IMPLEMENTATION AND RESULTS

The state machine bubble diagram in the below fig 1 shows the operation of a five-state machine that reacts to a single input and matches all the patterns having a+**,** b+c**,** and c*d+.

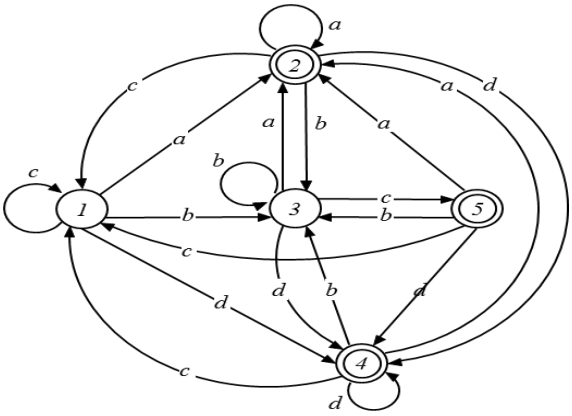

Fig 1:DFA1 for the pattern a+, b+c, and c*d+.

The set of literals are Σ=(a,b,c,d),Q=(1,2,3,4,5),q0=(1),F=(2,4,5).From the above DFA we can see that δ(acbd)=4, δ(bdca)=2, δ(bc)=5DFA1, XST has used eight flip flops for implementing the state machine and from Table 3 we can see that this schematic needs 17 macrocels,35 product term,21 function block,8 registers and 7 pins.

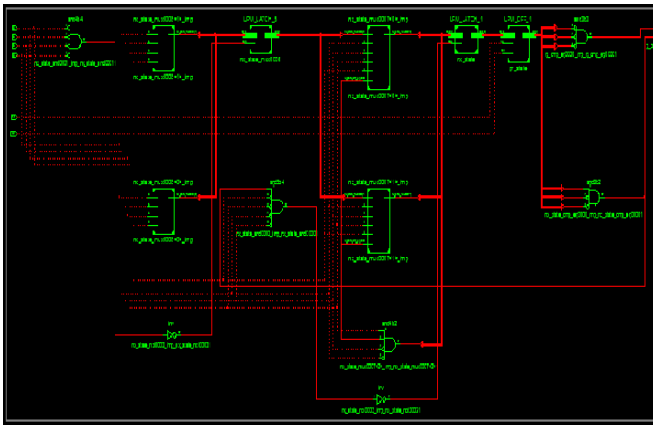

Fig2:technology schematic1 of the DFA1.

The equivalent D^2FA of the above DFA.The construction of D^2FA is done by eliminating the redundant transition and maintain the equivalence with the original DFA(the number of patterns accepted and rejected by the original DFA should not change)(fig3).So it is necessary to perform this reduction carefully. As you can see in above fig 2 by applying lemma 1 δ(1,c)= δ(2,c) so we can make a default transition from state 2 to state 1similarily other default transition if found by using the algorithm 3.4.2.Fig 3 shows the D^2FA of the Fig 1.

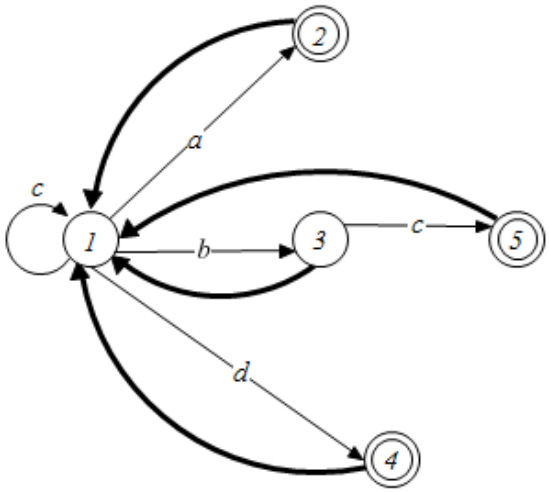

Fig 3:D^2FA,recognize the pattern a+, b+c, and c*d+

In fig 3 The set of literals are Σ=(a,b,c,d),Q=(1,2,3,4,5),q0=(1),F=(2,4,5). As you can see thatthe above DFA contain many number of default transition for eliminating the redundant transition so this reduces the space approximately 40% But the time taken to process the input is double or more then the original DFA because to match a single character two transitions needs to be taken so it increases the processing time The technology schematic is shown in fig 4.

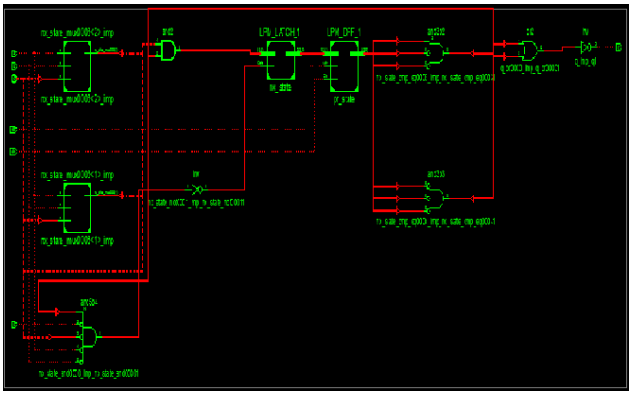

Fig4:technology schematic2 of the DFA2

As you can see from the schematic of DFA, XST the other resource summary of the technology schematic as we can see in table 2 the macrocels required is only 1 ,product term is 2,the functional blocks are 3,registers used is 1 and the total number of pins required are 5.

Table 2: Functional blocks used in DFA and D^2FA

-Function Blocks-	DFA	D^2FA
3-bit Register	1	1
#3-bit Latch	2	1
#IOs	7	7
# AND	16	11
# INV	27	23
# OR	6	2
# Flip-flops/Latches	8	6
# IO Buffers	7	7

As we can see from the table 2 that the resources used by the DDFA is very much less compare to the original DFA.So we can see that the area required is also reduces approximately 40% of the original.

Comparing the results of resources of DFA1 and D^2FA we can see that the macrocels required is 25% less for minimized DFA,product term used is 30 % less, function block required are 5%less ,registers used are 25 % less and pins used are also 30% less as the original DFA so overall we can conclude that the total area is very less as compare to the original DFA.

Table 3 shows the complete resource summary of the original DFA and D^2FA.We can see that the macrocells required is 25 % less,total product term used is 30 % less, functional block and registers used are 10% less and the pins required are 63% less to the original DFA in the implementation of the pattern that recognizes a+**,** b+c**,** and*c*d+.So we can conclude that the area reduced significantly in case of D^2FA.

Table 3:Resource summary of DFA and D^2FA

-Resources- ↓	DFA		D^2FA	
	Used/Total	**%**	**Used/Total**	**%**
Macrocells	17 /36	47%	12/36	34%
Product Terms	35 /180	19%	20/180	12%
Function Block	21 /108	19%	6/36	17%
Registers	8 /36	22%	7/34	21%
Pins	7 /34	21%	15/108	14%

4.1 Simulation Result:

Fig 5 presents the simulation result for pattern "bba" and fig 6 is the simulation waveform for pattern"bbabb".The graph can be easily interpreted .The first column(fig 5 and fig 6) shows the signal name it also shows the mode(direction)of the signals(the inward arrow shows the input and the outward arrow shows the output) .the second column shows the value of each signal in the position where the vertical cursor is placed(in fig 5 the cursor is at 710 ns and in this position the value of the output signal is 1 and all other are 0 similarly in fig 6 the cursor is at position 700 ns and in this position the value of the input signal b and output signal is 1 and all other are 0.The third column shows the simulation proper. The simulation waveform is same for the DFA and D^2FA but the clock pulse will be different.

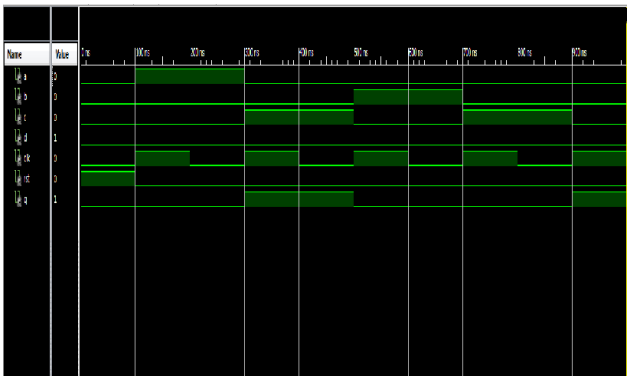

Fig5:Simulated Waveform of pattern "acbc"

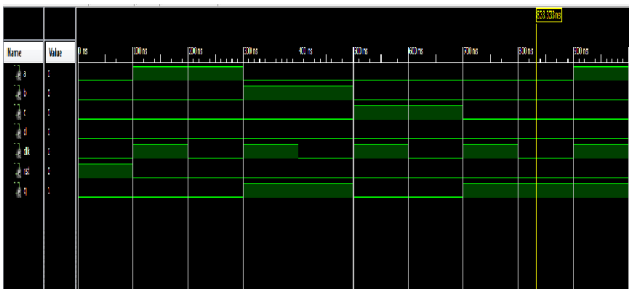

Fig6:Simulated Waveform of pattern "abca"

4.2 PerformanceAnalysis

Table 4 Here presents the performance summary with the comparison of Deterministic Finite Automata and delayed input deterministic finite automata(D^2FA). We can see the difference of different clock period and memory usage the total real time to completion in DFA is 6 sec whereas in D^2FA for the same is 7 secs and the total CPU time to completion in case of DFA is 5.77 secs whereas in D^2FA is 6.60 secs.So from this table we can conclude that the time needed for the completion in D^2FA is more than the original DFA.

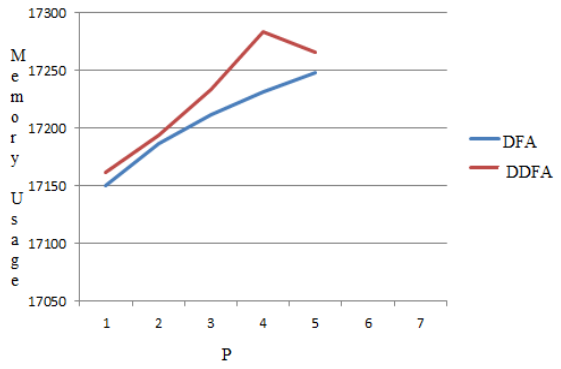

Fig 7: memory size for the different value of p(simulation result)

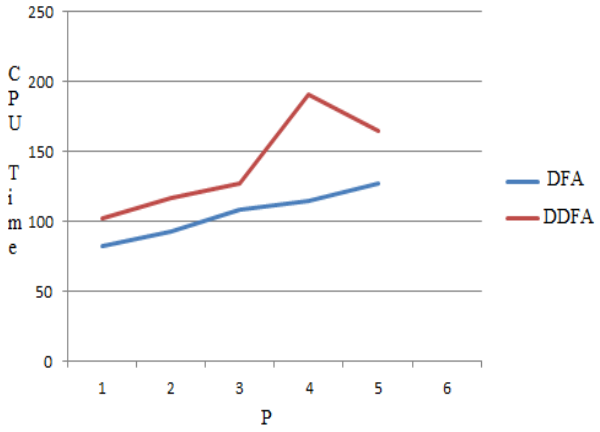

Fig 8: CPU time for the different value of p(simulation result)

Fig 7 and fig 8 shows the comparison graph for the DFA with the D^2FA Fig 7 shows the memory size required to process the input in KB for the different value of p (p is the pattern size) simulated is Xilinx 12.4 so you can compare memory usage in KB . The smallest pattern (p=1) requires 17151KB memory usage in case of DFA and 17168 KB memory usage in case of D^2FA.The smallest pattern (p=1)requires 94 ms in case of DFA and 104ms in case of D^2FA.Similarily the graph shows the memory required and CPU time for the patter size(p=1,2,3,4,5….) so we can conclude that the time taken to process any patterns needed more time than the original DFA because we needs to take extra default transition for processing. similarly fig 8 illustrate the CPU simulation time in ms for the different size of pattern p.So from these two graphs we can conclude that the both memory usage and CPU time is high in case of D^2FAcompare to the original DFA during the pattern matching.

Table 3:Performance summary

-Performance Factor-	DFA	D^2FA
Total REAL time to Xst completion	6.00 secs	7.00 secs
Total CPU time to Xst completion	5.77 secs	6.60 secs
Total memory usage	235256kb	231548kb

5. CONCLUSION

In this paper delayed input DFA is implemented for pattern matching which results the reduced area,but the increased number of clock pulse to process the inputpattern compare to the original DFA. In general the number of states required to implement both DFA and D^2FA is same but the number of transition edges are different.Number of resources used for the implementation reducedup to 60% of the original but the time and memory usage for matching pattern increases to 40% of the original. So the implementation in hardware with delayed input DFA is very apparent but the processing time is also very important factor so in future some techniques needs to be used to reduce the processing time of input pattern by reducing the total number of default edge transition.

7. REFERENCES:

[1] A. BabuKaruppiah, Dr. S Rajaram "Deterministic Finite Automata for Pattern Matching in FPGA for intrusion Detection" in International Conference on Computer and Electrical Technoogy-ICCCET 2011,18th& 19th March,2011."

[2] Jan Kastil, Jan Korenek Hardware Accelerated Pattern Matching Based onDeterministic Finite Automata with Perfect Hashing,IEEE 2010,p-149-152.

[3] Kai Wang, Yaxuan Q, YiboXue, Jun LReorganized and Compact DFA for EfficientRegular Expression Matching,IEEE communication society

[4] Hiroki Nakahara,TsutomuSasao and Munehiromatsuura "A Regular Expression Matching Using Non-Deterministic Finite Automata" in IEEE 2010.

[5]IvanoBonesana,MarcoPaolieri,Marco D. Santambrogio "An adaptable FPGA based system for regular expression Matching" in IEEE 2008.

[6] Mother Aldwairi,ThomasConte,PaulFranzon "Configurable string Matching Hardware for Speeding up Intrusion detection" inACM SIGARCH Computer Architecture News in,Vol. 33,No. 1, March 2005.

[7]Ashok kumarTummala and ParimalPatel"Distributed IDS using Reconfigurable Hardware" in IEEE 2007.

[8]Hoang Le and Viktor K. Prasanna Ming Hsieh Department of Electrical Engineering University of Southern California Los Angeles, CA 90089, USA A Memory-Efficient and Modular Approach for String Matching on FPGAs ,2010

[9]M. Dhanapriya,C. Vasanthanayaki "Hardware Based Pattern Matching Technique for Packet Inspection of High Speed Network" International Conference on "Control,Automation,Communication and energy

Consevation-2009 4th -6thjune 2009.

[10] IoannisSourdis, DionisiosN.Pnevmatikatos, and StamatisVassiladis," Scalable Multigigabit Pattern Matching for Packet Inspection," in Proc. IEEE Symp.Field program.Custom Comput. Feb. 2008.

[11] B. L. Hutchings and R. Franklin and D. Carver "Scalable hardware implementation usonf Finite Automata"Department of Electrical and Computer Engineering.

[13] J. Hasan, S. Cadambi, V. Jakkula and S. Chakradhar, "Chisel: A Storage-efficient, Collision-free Hash-based Network Processing Architecture," 33rd International Symposium on Computer Architecture,p.203-215.

[14]ReetinderSidhu,Vikot K. Prasanna "Fast Regular Expression Matching using FPGAs"9th Annual Symposium IEEE2001.

[15] Sailesh Kumar, SarangDharmapurikar, Fang Yu, Patrick Crowley, Jonathan Turner "Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection", *SIGCOMM'06*, September 11-15, 2006, Pisa, Italy, 2006 ACM.